A Hybrid Static–Dynamic Deep Learning Approach for Malware Classification

[1]Yash Agarwal*, [2]Shikhar Srivastava, [3]Sanket Jain, [4]Nisha Pal, [5]Sanjay Khakhil

[1, 2, 3, 4, 5] Galgotias College of Engineering and Technology (affiliated to Dr A. P. J. Abdul Kalam Technical University, Lucknow, India)
[1]12.yashag@gmail.com
[2]shikhar6055@gmail.com
[3]sanketjain4mar@gmail.com
[4]nisha.mnnit@gmail.com
[5]Skhakhil@gmail.com

Abstract. The rapid proliferation of malware and the sophistication of evasion strategies have diminished the effectiveness of traditional signature-based detection approaches. While static analysis is computationally efficient, its susceptibility to evasion remains a challenge; dynamic analysis offers behavioral knowledge at a higher computational cost. In this paper, we introduce a hybrid deep learning model that combines Printable String Information (PSI) derived from static analysis with dynamic API call sequences to improve malware classification accuracy. We use a multi-input neural network model to classify malware using the proposed feature set. We evaluated our proposed model using the BODMAS dataset and found that our proposed model achieves an accuracy of 98.32%, which is superior to individual static (92.4%) and dynamic (94.1%) models.

Keywords: Malware Detection, Hybrid Analysis, API Call Analysis.

1 Introduction

As the digital age grows, so does the need to get online. Managing money on the phone or computer, saving documents online, and working from home – all of these things have people relying heavily on the internet. Of course, all these developments make things go faster and easier; however, they also provide scammers an opportunity to slip in, especially with nasty software that can cause problems. Malware, short for malicious software, refers to any app designed with the intent to disrupt devices, steal data, or infiltrate systems without the user's consent. Based on their behavior and transmission, malware is classified into categories such as bugs that replicate themselves, self-propagating codes, hidden traps, snooping tools, multi-layered latent representations, and ransomware-based extortion. The ever-changing landscape of malicious technology sees new variants emerge daily while others undergo transformations to evade detection [1]. The scale of this issue has grown quickly in the last decade. The AV-TEST Institute registers over 450,000 new malicious programs daily. The reason why hackers are producing malware at such a rapid pace is that it is no longer possible to manually scan all of it – hence the use of auto-tools [2]. However, to address this problem, many antivirus software programs use signature-based detection. This software relies

B. Singh et al. (eds.), *Proceedings of the International Conference on Advances in Computing Technology and Artificial Intelligence (COMPUTATIA 2026)*, Atlantis Highlights in Intelligent Systems 18,
https://doi.org/10.2991/978-94-6239-713-2_21

on the scanning of files against a pre-known pattern of malicious attributes. Although this is effective in detecting known malware, this approach encounters a significant problem in that it is unable to detect new or modified malware unless the pattern list is regularly updated. This explains why these software programs often fail to detect zero-day attacks or changing viruses [3], [4].

Malware scans can be categorized into two types: static or dynamic. Static approaches don't run the file but analyze it by extracting pieces of information such as opcodes, function calls, layout maps, or text. Although this information provides insight into how the program works internally, it can be easily fooled by scrambled code, compressed malware, or modified scripts that are used in modern malware [5], [7]. Meanwhile, dynamic analysis observes the behavior of the program as it executes. It monitors activities such as system calls, file modifications, and changes in configuration, but also network communications. Because it observes what happens in real time, rather than just looking at source code, tricks that obfuscate code become less important in this case. However, resource requirements are high, and these methods require isolated testing areas [8], but even then, some malware will evade detection by detecting sandbox environments or waiting out timers. However, since neither approach is entirely comprehensive on its own, a new trend in malware research has emerged, which involves a combination of both static and dynamic properties. This has enabled the development of systems that utilize machine learning algorithms to achieve better detection rates and are more resistant to malware evolution [9], [19].

This paper proposes a malware detection system that combines both static and dynamic characteristics and uses machine learning algorithms for automatic classification. The aim is to improve the detection accuracy, minimize the false positives, and improve the system's capability to detect unknown malware samples. This means that attackers employ tactics such as code-shaping manipulation to conceal malicious software. As a result, traditional malware detection tools are rendered ineffective. This has led online protection companies to employ new approaches, such as identifying unusual behavior rather than file patterns. The new approaches monitor how programs behave, meaning that they are able to detect malicious software even if it is unknown. With enough data to analyze, intelligent systems are able to identify unusual patterns that traditional filters ignore [9], [20].

2 Related Work

A comprehensive body of research has investigated malware classification using both static and dynamic feature sets. Static analysis examines the code without execution, while dynamic analysis monitors behavior during runtime. This section highlights notable contributions in this domain, focusing on how different feature extraction and classification strategies have evolved. Salehi introduced a sophisticated detection technique centered on Windows API calls, as shown in Table 1, and their specific arguments. Rather than simply logging which APIs were called, they analyzed the arguments passed to these functions to gain deeper understanding of the intent of the code. By combining these API features with argument analysis and applying rigorous feature

selection methods to reduce dimensionality, they demonstrated significantly improved accuracy. Their experiments concluded that the Random Forest classifier yielded the best results, effectively handling the complex decision boundaries inherent in the data [7]. Taking a frequency-centric approach, Tian proposed a behavioral malware detection method that uses API occurrence analysis. In their framework, APIs were selected as features only if their appearance frequency exceeded a predefined threshold, ensuring that only the most relevant behaviors were captured. The resulting feature vectors were constructed as binary indicators that strictly reflect the presence or absence of specific API calls. Their experiments on a diverse dataset of malware and benign samples reported promising classification performance, highlighting the efficiency of frequency based filtering [6].

Islam sought to bridge the gap between analysis types by developing a hybrid system. Their approach combined static attributes, particularly the function length distribution and printable strings, with dynamic execution traces, such as API calls and their parameters. Instead of treating these as separate data streams, these distinct features were merged via vector concatenation (x = [xs; xd]) into a unified feature vector. This combined learning approach allowed the classifier to leverage the structural consistency of static features alongside the behavioral accuracy of dynamic traces, thereby improving overall classification accuracy [19]. In the realm of structural analysis, Park introduced a graph based detection framework utilizing a Kernel Object Behavioral Graph (KOBG). Their methodology moved beyond simple lists of actions by extracting malware family patterns using a mining approach to create a Weighted Common Behavioral Graph (WCBG). This method was particularly effective for identifying malware variants; by utilizing graph matching techniques, the system could detect malicious code that had been slightly altered but retained the same underlying behavioral structure [17]. Liangboonprakong and Sornil focused on the raw binary structure, presenting an N-gram based static classification approach. In their study, executable files were converted into hexadecimal sequences. From these raw sequences, they employed sequential pattern mining algorithms to extract frequent patterns that served as signatures. These patterns formed a robust feature set for classification, achieving strong identification results without the overhead of running the malware [5].

Table 1. Some Malicious API Patterns

Malicious activity	**API pattern**
Keylogger	(FindWindowA, ShowWindow, GetAsyncKeyState) (Set Windows HookEx, RegisterHotKey, GetMessage, and UnhookWindowsHookEx)
Anti-debugging	(IsDebuggerPresent, CheckRemoteDebuggerPresent, OutputDebugStringA, OutputDebugStringW)
Downloader	URL Download To File (Win Exec, Shell Execute)
DLL injection	OpenProcess, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread Operability
Dropper	FindResource, LoadResource, SizeOfResource

IAT hooking	LoadLibrary, (strcmp, strncmp, stricmp, strnicmp), VirtualProtect
Encryption	CryptCreateHash, CryptHashData, CryptGetHashParam, CryptAcquireContextA
Screen capture	(Get DC, Get Window DC), create a compatible DC, create a compatible bitmap, Select Object, Bit Blt, Write File
Registry behavior	RegSetValueExA, RegEnumValueA, RegDeleteKeyA, RegCreateKeyExA, OpenProcessToken

3 Methodology

This section discusses the design and implementation of the proposed hybrid malware detection framework. The approach combines static and dynamic analysis to leverage the structural information of executable files as well as the runtime behavior of the files during execution. The framework consists of four major phases: data acquisition, feature extraction, feature integration, and deep learning classification [10].

3.1 Architecture of the Proposed Method

The system is designed as a parallel processing framework as shown in Fig. 1, where static and dynamic components contribute complementary information. Static analysis extracts Printable String Information (PSI) from the binary, while dynamic analysis extracts API call sequences from execution logs. For example, In the system architecture, these features are processed by distinct neural network branches before being fused into final Classification.

3.2 Feature Extraction

Static Features: Static characteristics are pulled off without executing the code. With the use of the strings tool, readable text is pulled off from the file, and we remove anything that is shorter than nine letters long to remove clutter. The remaining string becomes part of a data array. The values in the array are scaled equally with a scaler to align them.
Dynamic Features: Dynamic Features: Behavioral sequences are extracted from execution logs.

3.3 Deep Learning Model Architectures

To evaluate the efficacy of the proposed features, we implemented three distinct deep learning models using the TensorFlow/Keras framework.

- **Static Model (MLP):** The static model is a Multi-Layer Perceptron (MLP) network. The input is a standardized vector of static features. The network

architecture includes a dense layer with 256 neurons (ReLU activation function), followed by a layer of batch Normalization for stabilizing the training process. To avoid overfitting, a dropout layer (rate=0.3) is added, succeeded by additional dense layers with 128 and 64 neurons.

- **Dynamic Model (Bi-LSTM).** The dynamic model employs a Bidirectional Long Short-Term Memory (Bi-LSTM) network to handle the sequential API data. The input sequence is fed through an embedding layer with a dimension of 128. The main processing component is a bidirectional LSTM layer with 128 units, enabling the network to learn forward and backward dependencies. This is followed by a dropout layer with a rate of 0.4 and a dense layer with 64 units.
- **Hybrid Model**. The proposed hybrid model uses a multi-input architecture. As detailed in the architecture section, the model processes static and dynamic inputs together. Through separate branches. The latent representations from both branches are concatenated into a single vector, which is passed through a final dense classification layer.

4 Algorithm

Require: Static Features Xs, Dynamic API Sequences Xq, Labels Y
Ensure: Trained Hybrid Classification Model M

1. Preprocessing:
2. Split Xs, Xq, and Y into training (80%) and testing (20%) sets.
3. Scale Xs using StandardScaler.
4. Define Static Branch (BS):
5. Input Is ← Shape of Xs.
6. Ls1 ← Dense(128, ReLU) applied to Is;
7. Ls2 ← Dropout(0.3) applied to Ls1;
8. Output Os ← Dense(64, ReLU) applied to Ls2.
9. Define Dynamic Branch (Bd):
10. Input ID ← Sequence Length of Xq;
11. E ← Embedding(Vocab=5000, Dim=128) applied to ID;
12. Ld1 ← Bidirectional LSTM(128) applied to E; 13: Output Od ← Dense(64, ReLU) applied to Ld1: 14: Feature Fusion:
13. Fvector ← Concatenate(Os, Od);
14. Hlayer ← Dense(64, ReLU) applied to Fvector;
15. Prediction ← Dense(1, Sigmoid) applied to Hlayer;
16. Model Compilation:
17. Compile using Adam Optimizer ($1e^{-3}$) and Binary Crossentropy.
18. Fit model on (Xtrain, Xtrain, Ytrain);

5 Experimental Setup and Feature Extraction

5.1 Dataset and Environment

The experimental assessment was done using the dataset of 997 malicious executables and 490 harmless files. The samples were sourced from VirusTotal and VX-

Underground. On the other hand, harmless ones were collected from clean system installations. We have run the static analysis in an environment using Ubuntu 14.04. Dynamic analysis was executed in Cuckoo Sandbox installed on Ubuntu 10.04 LTS, which interfaces with virtual machines running Windows XP in VMware Workstation 10.0 for safe execution and monitoring of the binaries. We have used the Python implementation of deep learning using the libraries TensorFlow and Keras [11].

5.2 Feature Selection and Hyperparameters

For static analysis, 7,253 unique features were identified after preprocessing was completed. For dynamic analysis, we fixed a vocabulary size to 5,000 in the embedding layer to capture the most significant API calls. All models were trained for 20 epochs with a batch size of 128. All specific hyperparameters for deep learning models are given in Table 2.

5.3 Evaluation Results

The proposed method was tested using the standard classification metrics. As can be seen from the experimental results in Table 3, the maximum detection accuracy of 98.32% was obtained by the integrated hybrid method. This is significantly better than the individual static and dynamic models, thus proving that the combination of structural and behavioral features is more effective in defending against malware.

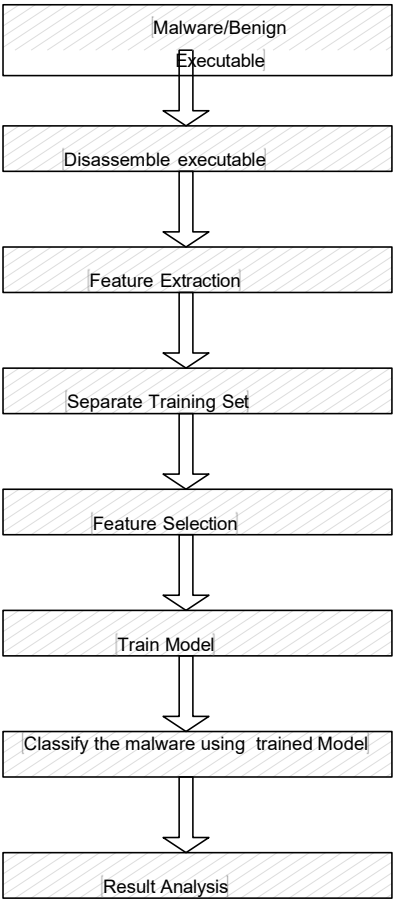

Fig. 1. Diagram of the Process Flow

Table 2. Deep LEARNING HYPERPARAMETERS

Parameter	**Value**
Optimizer	Adam
Learning Rate	1×10^{-3}
Loss Function	Binary Crossentropy
Batch Size	128
Epochs	20
Dropout Rate	0.3 (Static), 0.4 (Dynamic)
LSTM Units	128 (Bidirectional)
Embedding Dim	128

Table 3. Classification Results of Static, Dynamic and Integrated Methods

Method	TPR	FPR	Accuracy (%)
Static PSI Method	0.951	0.132	95.42
Dynamic API Call Grams	0.971	0.092	97.08
Integrated Hybrid Method	0.984	0.059	98.32

6 Conclusions

This paper proposes a combination of deep learning techniques for malware detection by making use of fixed text patterns as well as real-time API calls. The results clearly indicate that combining both disparate data types in a single model provides a 4.22% accuracy improvement over standalone dynamic models. These outcomes suggest that the bidirectional LSTM module is effective in identifying time-related patterns in API calls, while the fixed-pattern MLP side performs better in identifying unusual structural patterns. This complementary combination reached a peak accuracy of 98.32%, further emphasizing the effectiveness of combining different approaches in achieving better results when dealing with stealthy malicious attacks. In our future works, we will concentrate on improving the computational efficiency of deep learning models for real-time detection on devices with low computational power. Attention mechanisms [16], [18] are also explored in our future plans to emphasize more the most important sequences of API calls within the dynamic execution logs. Furthermore, evaluating the robustness of our model against adversarial evasion attacks [14], [15] and integrating explainable AI tools to interpret model predictions [12], [13] remains a priority for future work.

References

[1] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *Proc. 2010 Int. Conf. Broadband, Wireless Comput. Commun. Appl. (BWCCA)*, Fukuoka, Japan, 2010, pp. 297–300, doi: 10.1109/BWCCA.2010.85.
[2] Verizon, "2024 Data Breach Investigations Report (DBIR)," Verizon Enterprise, White Paper, 2024.
[3] H. Anderson and P. Roth, "EMBER: An open dataset for training static PE malware machine learning models," 2018, *arXiv:1804.04637*.
[4] E. Raff, J. Barker, J. Sylvester, R. Brooks, M. Intis, and P. Cardenas, "Malware detection by eating a whole EXE," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, 2018.
[5] J. Saxe and J. Berlin, "Deep neural network based malware detection using two-dimensional binary program features," in *Proc. 2015 10th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, 2015, pp. 11–20.

[6] Z. Yuan, Y. Lu, and Y. Xue, "DeepMal: Deep learning framework for Android malware detection," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defences (RAID)*, 2016.
[7] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Proc. Australasian Joint Conf. Artif. Intell.*, 2016, pp. 137–149.
[8] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware analysis via hardware virtualization extensions," in *Proc. 15th ACM Conf. Comput. Commun. Secur. (CCS)*, 2008, pp. 51–62.
[9] [1] M. S. Akhtar and T. Feng, "Evaluation of Machine Learning Algorithms for Malware Detection," *Sensors*, vol. 23, no. 2, p. 946, 2023. https://doi.org/10.3390/s23020946
[10] N. McLaughlin et al., "Deep Android malware detection," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, 2017, pp. 301–308.
[11] L. Yang, A. Ciptadi, I. Laziuk, A. Aliaga, and O. Chou, "BODMAS: An open dataset for learning based temporal analysis of PE malware," in *Proc. Deep Learn. Secur. Workshop (DLS)*, 2021.
[12] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in *Proc. Adv. Neural Inf. Process. Syst. 30 (NeurIPS)*, 2017.
[13] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?': Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1135–1144.
[14] C. Szegedy et al., "Intriguing properties of neural networks," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2014.
[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2015.
[16] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst. 30 (NeurIPS)*, 2017.
[17] L. Han et al., "GNN-based Android malware detection," in *Proc. 2020 IEEE Symp. Comput. Commun. (ISCC)*, 2020, pp. 1–6.
[18] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2017, pp. 1273–1282.
[19] P. V. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Comput. Sci.*, vol. 46, pp. 804–811, 2015.
[20] J. Kang and Y. Won, "A study on variant malware detection techniques using static and dynamic features," *J. Inf. Process. Syst.*, vol. 16, no. 4, pp. 882–895, 2020.